WO2002019076

Publication Title:

HYBRID PRIVILEGE ENFORCEMENT IN A RESTRICTED EXECUTION ENVIRONMENT

Abstract:

Abstract of WO 0219076

(A2) Translate this text A system and method for static and dynamic enforcement of access to resources through a runtime engine allows optimal selection of the enforcement method to improve performance, security, and maintainability of an execution environment. A trust state indicative of permitted resources is defined. Access to resources is provided by invoking particular function instantiations in the runtime engine. Binding of an executable entity to instantiations in the runtime engine occurs selectively during static enforcement based on the trust state. Runtime checks of the trust state by the executable entity occurs during dynamic enforcement. If the trust state does not correspond to a desired resource, access to that resource is prevented.

---

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
7 March 2002 (07.03.2002)

PCT

(10) International Publication Number
WO 02/19076 A2

(51) International Patent Classification[7]: G06F 1/00

(21) International Application Number: PCT/US01/41732

(22) International Filing Date: 15 August 2001 (15.08.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/229,481 31 August 2000 (31.08.2000) US
09/671,034 27 September 2000 (27.09.2000) US

(71) Applicant: CURL CORPORATION [US/US]; 8th Floor, 400 Technology Square, Cambridge, MA 02139 (US).

(72) Inventors: MICHAYLOV, Spiro; 47 Oak Hill Drive, Arlington, MA 02474 (US). MAZER, Murray, S.; 56 Robbins Road, Arlington, MA 02476 (US). KRANZ, David, A.; 115 High Haith Road, Arlington, MA 02476 (US).

(74) Agents: SMITH, James, M. et al.; Hamilton, Brook, Smith & Reynolds, P.C., 530 Virginia Road, P.O. Box 9133, Concord, MA 01742-9133 (US).

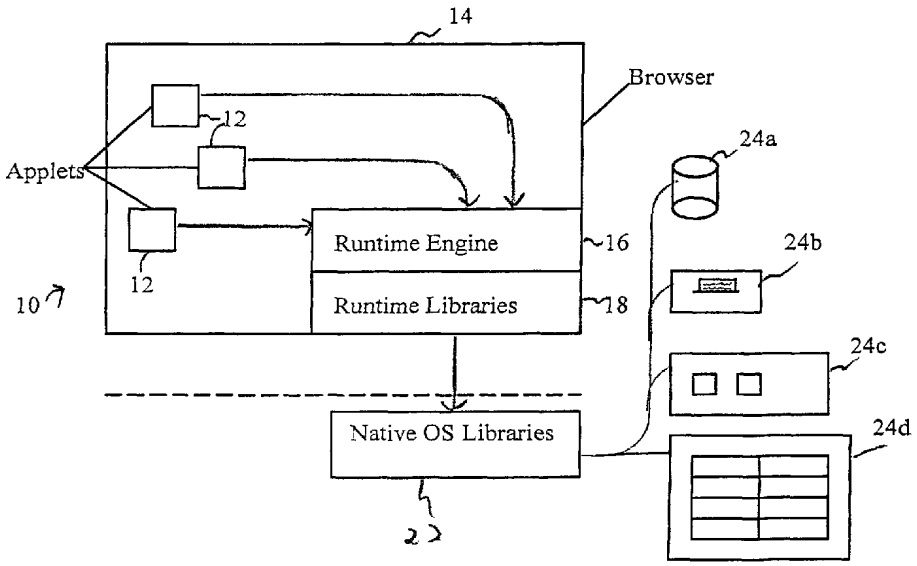
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

[Continued on next page]

(54) Title: HYBRID PRIVILEGE ENFORCEMENT IN A RESTRICTED EXECUTION ENVIRONMENT



(57) Abstract: A system and method for static and dynamic enforcement of access to resources through a runtime engine allows optimal selection of the enforcement method to improve performance, security, and maintainability of an execution environment. A trust state indicative of permitted resources is defined. Access to resources is provided by invoking particular function instantiations in the runtime engine. Binding of an executable entity to instantiations in the runtime engine occurs selectively during static enforcement based on the trust state. Runtime checks of the trust state by the executable entity occurs during dynamic enforcement. If the trust state does not correspond to a desired resource, access to that resource is prevented.

WO 02/19076 A2

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

HYBRID PRIVILEGE ENFORCEMENT IN A RESTRICTED EXECUTION ENVIRONMENT

BACKGROUND

In a computing execution environment, resources are typically accessed from many different executable entities. Improper access to a resource can have a negative result on the execution environment. Accordingly, access methods are known which enforce access restrictions to a resource. Typically a trust model is developed to designate which resources may be accessed by the executable entities. The trust model provides a framework to designate an executable entity as trusted or untrusted. If an executable entity is trusted, it will be permitted to access many or all resources. Similarly, if an executable entity is untrusted, it will be permitted to access few or no resources, or only permitted access in limited ways.

Systems which invoke mobile code in an executable entity such as an applet have a particular need for a trust model. Mobile code is code that is transported across a network before it is executed. Such mobile code is particularly useful in email and web page applications because these mediums allow the executable entity to easily propagate to other systems. Since the source of the mobile code may be unfamiliar, however, there is greater risk of a negative result from executing the mobile code.

Enforcement of access to resources according to the trust model is enabled by a trust indicator. The trust indicator indicates the level of trust afforded to an executable entity according to the trust model. For a particular resource, the trust indicator can be referenced to determine if access to that resource should be permitted or denied.

Known platforms which may employ a trust model to enforce access restrictions computation include Java™, marketed commercially by Sun Microsystems of Palo Alto, California, and Tcl and Safe-Tcl by Ajuba Solutions of Mountain View, California.

Enforcement can be static or dynamic. Static enforcement occurs prior to runtime of the executable entity when external references of the executable entity are associated with, or bound, to a particular instantiation in the runtime engine. Dynamic enforcement occurs during execution, after the external references in the executable entity have been bound to a particular instantiation in the runtime engine. Therefore, static enforcement examines the trust indicator during binding, and thus allows selective binding based on the trust indicator. Dynamic enforcement examines the trust indicator during runtime, and determines whether access to the resource should be permitted or denied.

SUMMARY

A method of hybrid enforcement in a restricted execution environment having protected resources implements a trust model having a plurality of trust levels. Hybrid enforcement includes static and dynamic enforcement, and includes defining a trust state indicative of the trust levels and executing an executable entity adapted to invoke a runtime engine operable to access the protected resources. Enforcement according to the trust model then occurs by protecting, via dynamic enforcement, access the resources and also protecting, via static enforcement, access to the resources, wherein static enforcement and the dynamic enforcement occur with respect to the same executable entity.

Depending on the context in which the resource is accessed, one type of enforcement may be more advantageous than another. Dynamic enforcement increases runtime overhead, as additional instructions are employed to examine the trust indicator during runtime. However, static enforcement must occur at entry point call where a resource is accessed, and therefore may be difficult to implement and verify in a large system with many entry point calls to the resource, such as via complex indirect paths. Typically, however, an execution environment is designed

to implement a particular type of enforcement. Accordingly, it is difficult to select an optimal enforcement mechanism for all resource access contexts.

In accordance with the present invention, static and dynamic enforcement of access to resources through a runtime engine allows optimal selection of the enforcement method to improve performance, security, and maintainability of an execution environment. A trust state indicative of permitted resources is defined. Access to resources is provided by invoking particular function instantiations in the runtime engine. Binding of an executable entity to instantiations in the runtime engine occurs selectively during static enforcement based on the trust state. Runtime checks of the trust state by the executable entity occurs during dynamic enforcement. If the trust state does not correspond to a desired resource, access to that resource is prevented.

Protection of a resource via static enforcement includes compiling the code defining the executable entity against packages which contain only functions that allow access to permitted resources, either by omitting privileged functions, by stubbing references to privileged functions, or by selecting particular instantiations from among the packages. Protection of a resource via dynamic enforcement includes binding the executable entity to an instantiation of a function operable to examine the trust state at runtime, and disallowing access to the resource by branching around the access to the resource if the trust state does not correspond to the resource. When implementing an execution environment, the developer may elect static or dynamic enforcement on a per-resource basis, and may alternate between static and dynamic enforcement in the same execution environment to employ the optimal enforcement method in each context.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other objects, features and advantages of the invention will be apparent from the following more particular description of preferred embodiments of the invention, as illustrated in the accompanying drawings. The drawings are not necessarily to scale, emphasis instead being placed upon illustrating the principles of the invention.

Fig. 1 shows a block diagram of a browser having applets executing in a runtime environment with a runtime engine adapted to call a runtime library which in turn calls native operating system (OS) libraries;

Fig. 2 shows an executable entity having functions associated with a runtime engine via external references;

Fig. 3a shows a block diagram of a calling structure of trusted and untrusted applets employing static enforcement in which each applet is compiled against an explicit, extended compilation package having references to the allowed functions;

Fig. 3b shows a chart of static enforcement techniques in which the compile time environment is controlled to include a) one or more incremental packages having explicit references to functions; b) one of several packages having explicit references to permitted functions but omitting references to disallowed functions; and c) a package having implicit references to allowed functions which inserts code stubs for disallowed functions;

Fig. 4a shows a block diagram of a calling structure of trusted and untrusted applets compiled against a package having multiple instantiations of the write function adapted to implement a runtime check on the write parameters to ensure only allowed parameters for the trust level are being passed, thus illustrating the dynamic, or runtime, component of static enforcement;

Fig. 4b shows a pseudocode implementation of a benign write routine;

Fig. 4c shows a pseudocode implementation of a privileged write routine;

Fig. 5 shows a pseudocode example of dynamic enforcement;

Fig. 6 shows a flowchart of the loading and execution of an executable entity employing static and dynamic enforcement;

Fig. 7a shows a call graph with a high fan-out, and therefore likely adapted for static enforcement; and

Fig. 7b shows a call graph with a high fan-in, and therefore likely adapted for dynamic enforcement.

DETAILED DESCRIPTION

A description of preferred embodiments of the invention follows.

In an execution environment, an access control mechanism is often implemented to prevent undesired or unintentional access to resources by executable entities in the execution environment. Undesired and unintentional access activity can have a negative effect, damaging or destroying data, hindering performance, or preventing system use. Such activity may be the result of user error, ignorance, or malice. Accordingly, a trust level is assigned to each executable entity based on an estimation of the likelihood of the executable entity performing negative access activity. Access to resources is permitted selectively based on the trust level. Typically the trust level is defined by a trust model which is designed to minimize the risk of negative effects without hindering system use.

Fig. 1 shows a system suitable for performing hybrid enforcement as defined herein. Referring to Fig. 1, an execution environment 10 has a plurality of executable entities 12, such as applets, executing in conjunction with an application such as a browser 14. The executable entities access resources via a runtime engine 16, which in turn invokes a runtime library of predefined functions 18. The runtime library 18 performs calls into the native operating system (OS) libraries 22 operable to access resources 24 generally, including disks 24a, printers 24b, routers 24c, mass storage devices 24d, and others.

In an executable entity, a calling structure is defined by the manner in which functions call other functions. Fig. 2 shows an executable entity and the calling structure to functions. Some functions are internal functions 26, and correspond to code contained within the executable entity 12. Internal functions 26 can call other internal functions 26. Internal functions may also make calls to external functions. External functions are functions corresponding to code not contained within the executable entity 12, such as functions in the runtime engine 16. When an internal function 26 calls an external function, an external reference 28 is generated in the executable entity 12. The external reference 28 allows access to an entry point to the external function. Prior to execution of the executable entity 12, external references must be associated, or bound, to an actual entry point of the code corresponding to the external function in the runtime engine 16. During execution, or runtime, a call

to the external function will result in control being transferred to the entry point of the external function code via the external reference 28.

In an executable entity 12, a trust indicator 30 can be referenced to determine the trust level of the executable entity. Reference to the trust indicator 30 can occur during binding, as external references are bound to entry points in the runtime engine, or during runtime, as the trust indicator is referenced when an already bound entry point is invoked for execution. The former is static enforcement. The latter is dynamic enforcement. Note that the trust indicator need not be physically stored in the same location.

In static enforcement, the trust indicator is examined during binding so that an external reference may be selectively bound to a particular entry point based on the trust level. If the trust level does not correspond to the function entered at the particular entry point, then the external reference will be stubbed around, omitted entirely from the entry points available for binding, or bound to an alternate, benign instantiation of the function consistent with the trust level.

In dynamic enforcement, an external reference is bound to the corresponding entry point without reference to the trust indicator at binding time. Conditional runtime checks in the runtime engine examine the trust level upon invocation of the function via the entry point. Therefore, each external reference to a particular function will be bound to the same entry point, but the code within the function accessed via the entry point will disallow or redirect access if the trust level does not correspond to the function to be performed.

Before a discussion of the hybrid enforcement implementation as disclosed herein, it may be beneficial to discuss the advantages of hybrid enforcement. Hybrid enforcement allows the selection of the optimal enforcement method to be used for each access to a protected resource. Implementation of the optimal type of enforcement can improve performance, reliability, security, maintainability, or a combination of these. A system architecture which imposes a particular enforcement method does not allow the developer to elect static or dynamic enforcement on a per-resource basis permit the optimal method to be selected on a case by case basis.

Fig. 3a shows a calling structure employing static enforcement. Referring to Fig. 3a, a trusted executable entity 12t and an untrusted executable entity 12u are each compiled against a particular package in the compilation environment 32. Trusted executable entity12t is compiled against trusted compilation package 32t. Untrusted executable entity12u is compiled against the untrusted compilation package 32u. The trusted compilation package 32t IO-full, contains references to both the read 34 and write 36 functions. The untrusted compilation package 32u IO-basic contains only references to the read 34 functions, since read is a benign function but write, or delete, may be damaging. The references in the trusted compilation package 32t allow calls into the runtime engine 42 for both the read 38 and write 40 functions. These functions 38 and 40 in turn may call other functions in the runtime engine 42, including the native OS fetch and write 46 and 48, respectively. The untrusted compilation package is permitted calls only to the benign read 38 in the runtime engine, 42.

Referring in more detail to Fig. 3a, the actual runtime execution behavior may occur differently according to several alternate embodiments. In Fig. 3a, the untrusted compilation environment 32u does not contain any code for calling the write function 40 in the runtime engine. The write call 50 in the untrusted executable entity 12u, therefore, would be undefined, as it does not reference a valid execution point. In one embodiment, the executable entity would not be permitted to begin execution until all external references are resolved. In another embodiment, the write call would be bound to a benign stub so that execution may continue. Such a stub may perform no action, or may print a message indicating that there are insufficient privileges to execute the write function, or may terminate execution due to insufficient privileges. In another embodiment, the write call would be left unbound, thus resulting in an attempt to execute an invalid reference and likely causing ungraceful termination. The chart shown in Fig. 3b shows the alternate embodiments used to bind the external references.

Referring to the chart of Fig. 3b, three levels of trust attributable to an executable entity are shown in column 52. Any number of levels of trust could be employed, depending upon the granularity of access control required. Explicit,

incremental binding, shown in column 54, involves binding against one or more packages, wherein each package includes functions requiring an additional level of privilege to execute. An executable entity will therefore not have access to entry points that do not correspond to the level of privilege of the executable entity. Explicit, extended binding, shown in column 56, binds each executable entity to one of several packages, wherein each of the packages of a higher privilege level includes all functions of lesser privileged packages in addition to the functions added for the current level. Implicit binding, shown in column 58, includes entry points for functions permitted at that level, and includes benign stubs for disallowed functions. In this manner, the packages available to bind against are controlled to allow access to functions through allowed entry points, and not including or including only benign stubs for disallowed functions.

Fig. 4a shows a particular embodiment in which different instantiations of a function are employed and selectively bound depending on the privilege level of the executable entity. A calling sequence for multiple instantiations of the write function are used as an example. This embodiment is similar to providing code stubs around disallowed functions, however, rather than binding to unintelligent stubs, runtime checks may be performed on the parameters passed to allow access only when the parameters passed indicate benign access, rather than disallowing access to the function altogether. Thus, there is a runtime component to this static enforcement. For example, a write function may be considered benign when a user is writing to the user's own directory. Accordingly, the untrusted instantiation of the write function would examine the directory that is to be written to, and disallow access only if the directory is not the user's directory. The trusted write would not require such a check. Note that the runtime checks are on the parameters, or data, to be acted upon and not on the trust state of the executable entity, thus illustrating the runtime component of static enforcement.

Referring to Fig. 4a, different compilation environments contained in packages 32t and 32u are used for both the trusted executable entity 12t and the untrusted executable entity 12u, respectively. The untrusted executable entity 12u, however, is bound to the Write1 instantiation 60, while the trusted executable entity

is bound to the Write2 instantiation 62. Fig. 4b shows the Write1 instantiation 60 and Fig. 4c shows the Write2 instantiation 62. Note that while the Write1 instantiation 60 includes a check at line 322 to ensure that the parameter indicating the directory to write to is the user's directory, the Write2 instantiation 62 requires no such check prior to calling the native Write function.

Fig. 5 shows a pseudocode example of dynamic enforcement in an executable entity. In dynamic enforcement, code is compiled without regard to the trust indicator, and a runtime check of the trust level of the executable entity is performed. Referring to Fig. 5, a function which performs dynamic enforcement is invoked to determine if access to a protected resource should be provided, as shown at line 300. The trust indicator corresponding to the trust level of the executable entity is retrieved, as depicted at line 302. The trust level required to access the protected resource in question is retrieved, as shown at line 304. In a particular embodiment, there are two levels of trust, however, other levels could be defined to map trust levels to various resources. The trust level of the executable entity is compared to the trust level required to access the resource, as disclosed at line 306. If the trust level of the executable entity corresponds to the trust level required for access to the resource, access_permitted is set to true, as depicted at line 308, and access to the resource is allowed. Otherwise, access_permitted is set to false, as shown at line 310, and access is disallowed. A message such as "Insufficient Privilege to Perform Function" may optionally be displayed, as well as termination of the executable entity. Since dynamic enforcement requires runtime computation and checking of the trust level for each invocation of the function as shown at line 306, performance can be reduced; however, other factors may also be considered, as will be described further below.

Fig. 6 shows a flowchart of the execution lifetime of an executable entity in a hybrid enforcement environment. Referring to Fig. 6, the applet code corresponding to the executable entity is downloaded to the client execution environment, as depicted at step 200. The applet code may be downloaded by any suitable method, such as via a web browser. The trust indicator corresponding to the applet code is examined to determine the trust level, as shown at step 202. The reference to the

trust indicator at step 202 corresponds to static enforcement, since the external symbols in the applet code have not yet been resolved. The symbol packages corresponding to the trust level are loaded so that external references may be resolved through compilation of the applet code, as disclosed at step 204. For each external symbol reference in the applet code, a lookup in the symbol packages is performed, as shown at step 206. A check is performed to determine if the external symbol was found in the symbol packages, as shown at step 208. If the external symbol was not found, than the applet code is not permitted to access the resource corresponding to the external reference, as shown at step 210. If the symbol is found, then the external reference is resolved with an entry point in the runtime engine, as shown at step 212. A check is performed to determine if there are any more external references in the applet code, as shown at step 216. If there are more symbols, the next external symbol reference is retrieved, as shown at step 205, and control reverts to step 206. If there are no more external symbol references, then compilation is complete and the static enforcement checks have all been implemented in the resulting executable entity.

Execution of the executable entity begins, as disclosed at step 218. The executable entity attempts to access a protected resource, as shown at step 220. The trust indicator of the executable entity is retrieved, as depicted at step 222, and compared to the trust level required to access the resource, as disclosed at step 224. This check corresponds to dynamic enforcement because the trust indicator is being checked conditionally at a branch into an already bound external reference. If the trust level of the executable entity corresponds to the trust level of the resource, then access is permitted by the executable entity, as shown at step 226. If the trust level does not correspond, then access is denied, as shown at step 228. Denial of access may terminate or disrupt execution. Control then reverts to step 220 upon subsequent attempts to access a protected resource.

As indicated above, there are several outcomes in the event of a trust indicator indicating an insufficient trust level. In the case of a symbol not found at step 208, the external reference may remain unresolved, possibly causing unpredictable operation or causing the executable entity to be unable to begin

execution. Alternatively, a benign stub may by bound, possibly printing an informative message as to why access was denied. Similarly, in the case of dynamic enforcement, denial of access may result in no action, in an informational message being displayed, or may terminate execution altogether. It is worth noting that regardless of the outcome when the trust indicator check indicates no access, the protected resource remains unaccessed by the untrusted executable entity.

As indicated above, hybrid enforcement allows selection of the optimal enforcement method for each context in which a function is invoked. Factors such as complexity, speed, fan in, fan out, the number of invocations, and others may be considered. Figs 7a and 7b below illustrate two call graph contexts, one of which is well adapted for dynamic enforcement, the other which is well adapted for static enforcement.

Referring to Fig. 7a, a call graph context adapted for dynamic enforcement is shown. Functions 100a-100n, in user code area 102, invoke functions 106a-106d, in the runtime library code area 104. Functions 106a-106d access the protected resource 112 via function 107. Due to the fan in from functions 106a-106d to function 107, a single runtime check in function 107 can provide effective protection of the resource 112, since function 107 is the only access point to the resource 112. Static enforcement, on the contrary, would require identifying all the calls into functions 106a-106d from the user code functions 100a-100n. As this is an illustrative example, it should be noted that a more complex call graph having many layers may occur. Since this method of enforcement is easily verifiable from a single check in function 107, security verification and maintainability are improved.

Referring to Fig. 7b, a call graph adapted for static enforcement is shown. Function 108 in the runtime engine area 104 is called from functions 100a-100n in the user code area 102. Function 108 is the only entry point which accesses the protected resource 112. Since the call graph which provides access to the protected resource is not complex, static enforcement corresponding to function 108 is well adapted to protect the resource 112. Static enforcement, therefore, is well adapted to protect a resource when the entry points which can access the resource are easily identifiable and enumerable. Complex call graphs, as illustrated in Fig. 7a, make it

difficult to verify that all call paths to the resource have been identified. Other design and development factors could similarly be employed in determining whether static or dynamic enforcement is well suited to a particular context.

Those skilled in the art should readily appreciate that the programs defining the operations and methods defined herein are deliverable to an execution environment in many forms, including but not limited to a) information permanently stored on non-writeable storage media such as ROM devices, b) information alterably stored on writeable storage media such as floppy disks, magnetic tapes, CDs, RAM devices, and other magnetic and optical media, or c) information conveyed to a computer through communication media, for example using baseband signaling or broadband signaling techniques, as in an electronic network such as the Internet or telephone modem lines. The operations and methods may be implemented in a software program executable out of a memory by a processor or as a set of instructions embedded in a carrier wave. Alternatively, the operations and methods may be embodied in whole or in part using hardware components, such as Application Specific Integrated Circuits (ASICs), state machines, controllers or other hardware components or devices, or a combination of hardware and software components.

While the system and method for hybrid enforcement have been particularly shown and described with references to embodiments thereof, it will be understood by those skilled in the art that various changes in form and details may be made therein without departing from the scope of the invention encompassed by the appended claims. Accordingly, the present invention is not intended to be limited except by the following claims.

CLAIMS

What is claimed is:

1. A method of hybrid enforcement in a restricted execution environment having at least one resource comprising:

establishing a trust model having a plurality of trust levels;

defining a trust state indicative of one of the trust levels;

executing an executable entity adapted to invoke a runtime engine operable to access the at least one resource;

protecting, via dynamic enforcement, access to at least one of the resources; and

protecting, via static enforcement, access to at least one of the resources, wherein the static enforcement and the dynamic enforcement occur with respect to the same executable entity.

2. The method of claim 1 wherein the static enforcement occurs prior to the executing.

3. The method of claim 2 wherein the static enforcement occurs via binding at least one external reference in the executable entity to a particular instantiation in the runtime engine.

4. The method of claim 3 wherein the binding includes resolution of symbols indicative of external references to an instantiation of a function.

5. The method of claim 4 wherein the dynamic enforcement occurs after binding the executable entity to a particular instantiation in the runtime engine.

6. The method of claim 1 wherein the static enforcement and the dynamic enforcement include referencing a trust indicator.

7. The method of claim 6 wherein the trust indicator is stored in a plurality of locations.

8. The method of claim 6 wherein the static enforcement includes selectively binding a particular instantiation in the runtime engine based on the trust indicator.

9. The method of claim 6 wherein the dynamic enforcement includes referencing the trust indicator during runtime.

10. The method of claim 1 wherein protecting via static enforcement includes a runtime check of a parameter.

11. The method of claim 1 wherein protecting via static enforcement includes selectively binding to a benign stub based on the trust state.

12. The method of claim 1 wherein the executable entity is an applet.

13. A system for hybrid privilege enforcement in an execution environment comprising:

an execution environment having at least one resource and operable to execute executable entities;

a runtime engine adapted to associate the executable entities with access to resources; and

a trust indicator adapted to indicate a trust level, wherein the trust level corresponds to which of the at least one resource is permitted to be accessed by the executable entity, and wherein the execution environment is operable:

to protect via static enforcement, the at least one resource via examining the trust indicator; and

to protect, via dynamic enforcement, the at least one resource via examining the trust state indicator, wherein the static enforcement and the dynamic enforcement occur with respect to the same executable entity.

14. The system of claim 13 wherein the trust indicator is adapted to be referenced during runtime of the executable entity.

15. The system of claim 13 wherein the executable entity further includes conditional instructions wherein the conditional instructions are executed selectively as a result of referencing the trust indicator.

16. The system of claim 13 wherein the execution environment further includes a linker operable to selectively associate the executable entity with the entry points as a result of the examining of the trust state indicator.

17. The system of claim 16 wherein the linker is further operable to selectively associate the executable entity with the entry points prior to runtime.

18. The system of claim 13 further comprising a runtime library including library code operable to access the resources.

19. The system of claim 13 wherein the runtime engine further includes entry points, wherein the executable entity is associated to the entry points.

20. The system of claim 19 wherein the entry points correspond to access to a resource.

21. The system of claim 19 wherein the executable entity further comprises external references, wherein the external references correspond to the entry points.

22. A computer program product including computer program code for hybrid enforcement in a restricted execution environment having at least one resource comprising:

computer program code for establishing a trust model having a plurality of trust levels;

computer program code for defining a trust state indicative of one of the trust levels;

computer program code for executing an executable entity adapted to invoke a runtime engine operable to access the at least one resource;

computer program code for protecting, via dynamic enforcement, access to at least one of the resources; and

computer program code for protecting, via static enforcement, access to at least one of the resources, wherein the static enforcement and the dynamic enforcement occur with respect to the same executable entity.

23. A computer data signal for hybrid enforcement in a restricted execution environment having at least one resource comprising:

program code for establishing a trust model having a plurality of trust levels;

program code for defining a trust state indicative of one of the trust levels;

program code for executing an executable entity adapted to invoke a runtime engine operable to access the at least one resource;

program code for protecting, via dynamic enforcement, access to at least one of the resources; and

program code for protecting, via static enforcement, access to at least one of the resources, wherein the static enforcement and the dynamic enforcement occur with respect to the same executable entity.

24. A system for hybrid privilege enforcement in an execution environment comprising:

means for establishing a trust model having a plurality of trust levels;

means for defining a trust state indicative of one of the trust levels;

means for executing an executable entity adapted to invoke a runtime engine operable to access the at least one resource;

means for protecting, via dynamic enforcement, access to at least one of the resources;

means for protecting, via static enforcement, access to at least one of the resources, wherein the static enforcement and the dynamic enforcement occur with respect to the same executable entity.

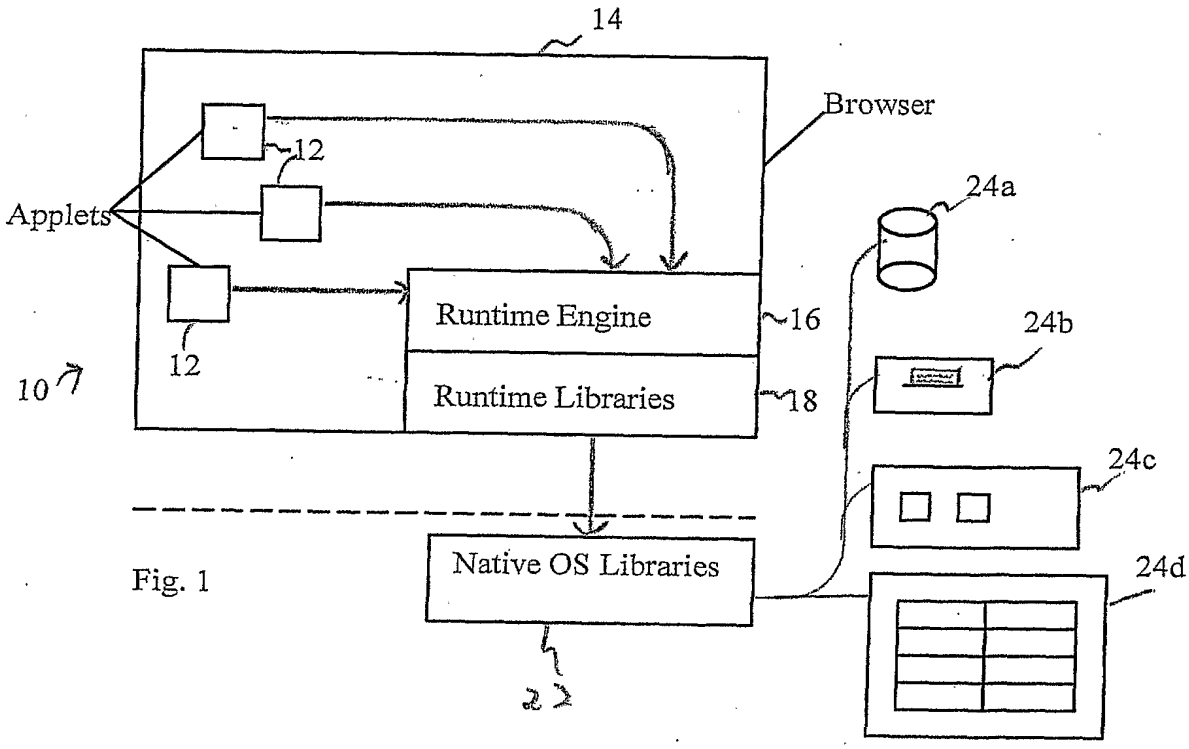

14
Browser
12
Applets
24a
Runtime Engine
16
24b
12
10
Runtime Libraries
18
24c
Native OS Libraries
24d
22

Fig. 1

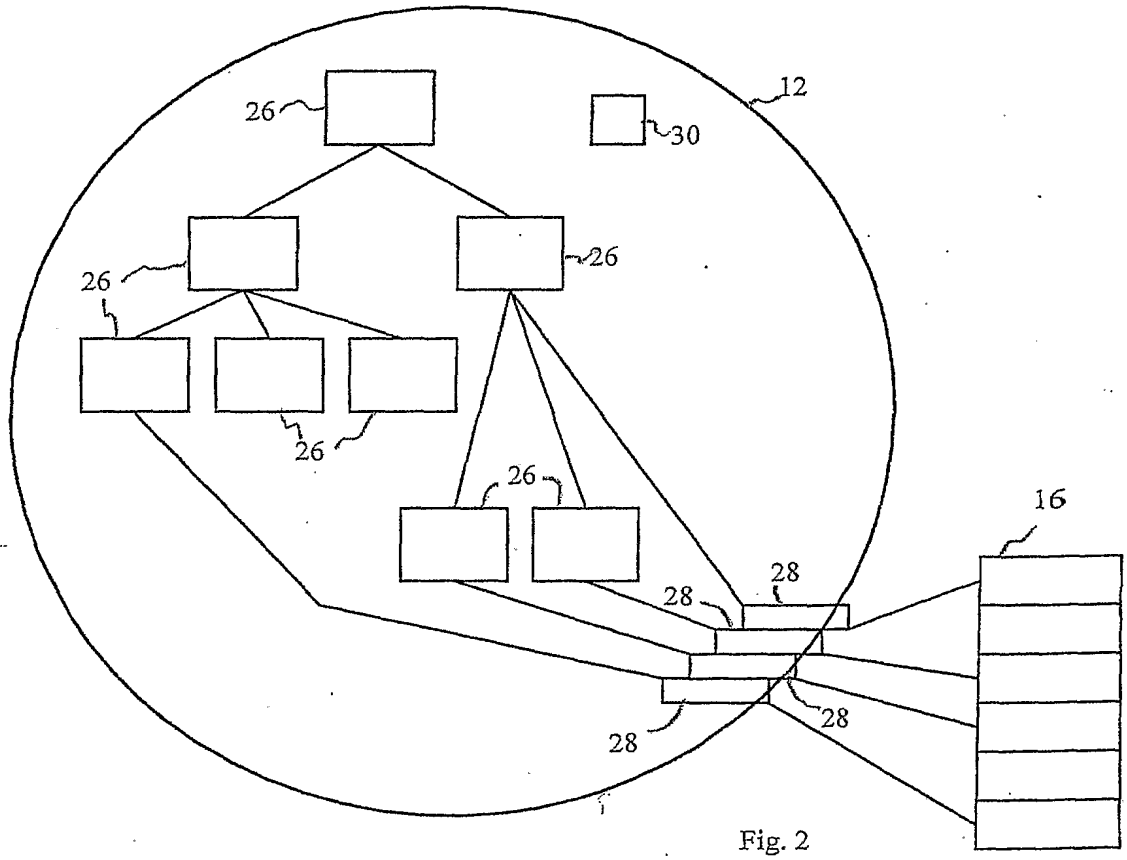

12
26
30
26
26
26
26
26
16
28
28
28
28

Fig. 2

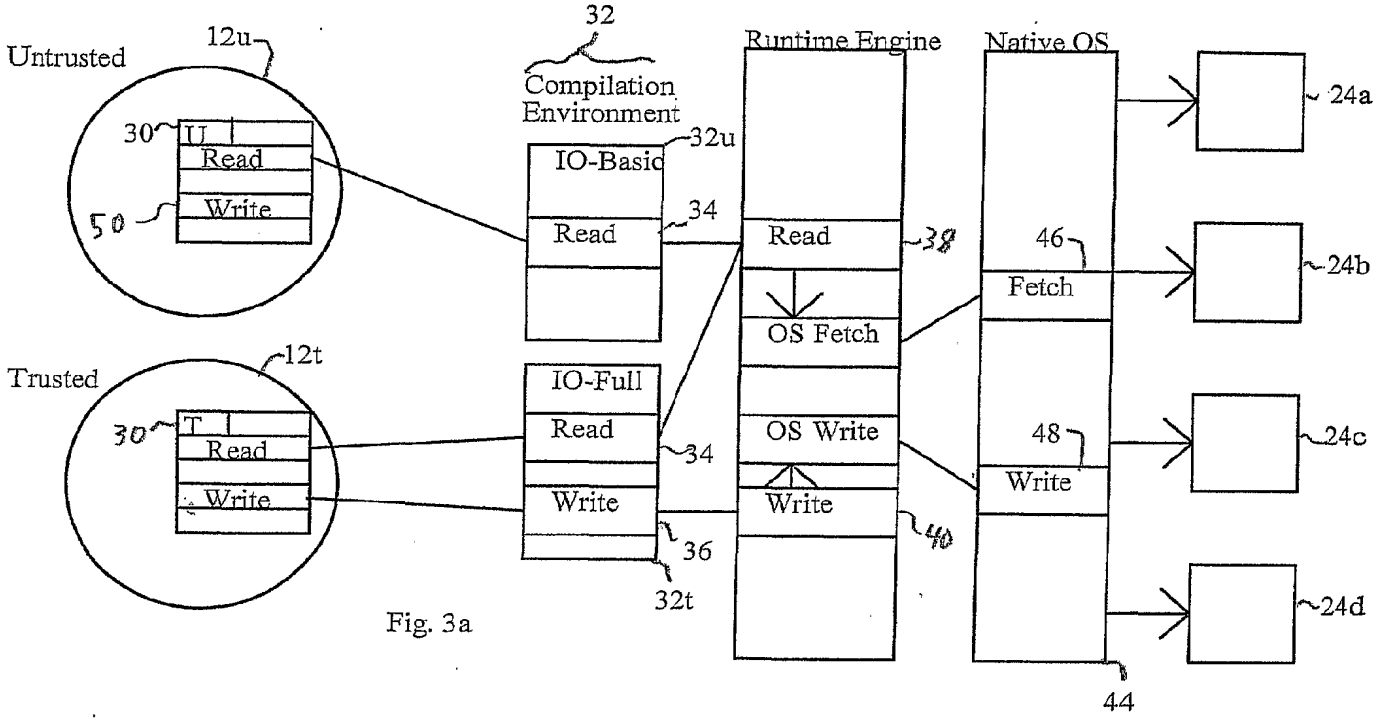


Fig. 3a

52

Trust level	54 Package Structure 56		58
	Explicit, Incremental	*Explicit, Extended*	*Implicit*
Untrusted	IO-BASIC Exists?: normal	IO-BASIC Exists?: normal	IO Exists?: normal Read: stub Write: stub
Semi trusted	IO-BASIC Exists?: normal IO-MEDIUM-EXTRAS Read: normal	IO-MEDIUM Exists?: normal Read: normal	IO Exists?: normal Read: normal Write: normal
Fully trusted	IO-BASIC Exists?: normal IO-MEDIUM-EXTRAS Read: normal IO-FULL-EXTRAS Write: normal	IO-FULL Exists?: normal Read: normal Write: normal	IO Exists?: normal Read: normal Write: normal

Fig 3b

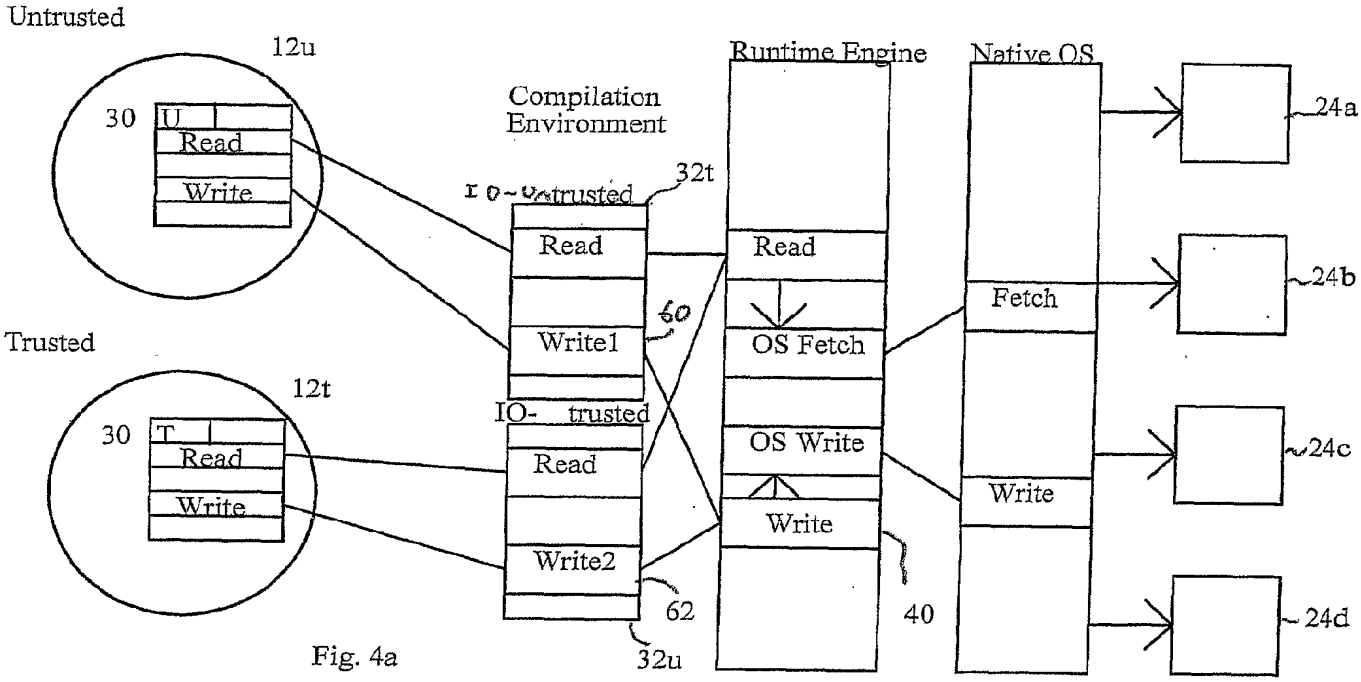

Untrusted
12u
30
U
Read
Write
Trusted
12t
30
T
Read
Write
Compilation Environment
IO-Untrusted
32t
Read
Write1
IO- trusted
Read
Write2
32u
60
62
Runtime Engine
Read
OS Fetch
OS Write
Write
40
Native OS
Fetch
Write
24a
24b
24c
24d

Fig. 4a

```
function write1 ( filespec )                                    ← 60

begin
        retrieve (user_home_directory)
        if ( dir ( filespec ) == user_home_directory)  ~~  322
        then
                write ( filespec )
        else
                throw_exception
end
```

Fig. 4b

```
function write2 ( filespec )                                    ← 62

begin
        write ( filespec )  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  330
end
```

Fig. 4c

```
function access_protected_resource (access_permitted)  ~~  300

begin
        retrieve (trust_indicator)  ~~~~~~~~~~~~~~~~~~~~~~~~~~  302
        retrieve (trust_level_required)  ~~~~~~~~~~~~~~~~~~~~~  304
        if (trust_indicator => trust_level_required)  ~~~~~~~~~  306
                access_permitted = true  ~~~~~~~~~~~~~~~~~~~~~~  308
        else
                access_permitted = false  ~~~~~~~~~~~~~~~~~~~~~  310
end
```

Fig. 5

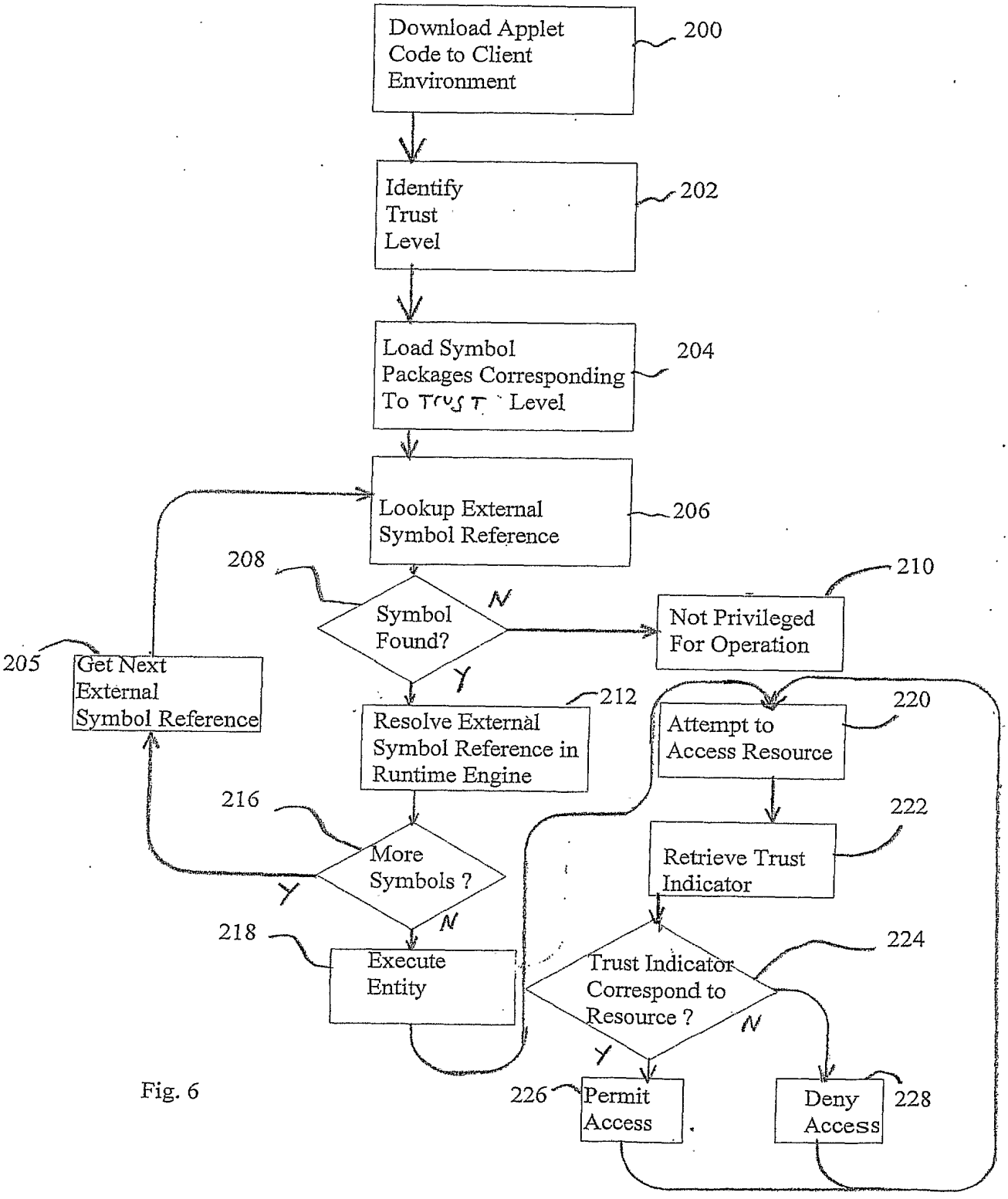

Download Applet Code to Client Environment
200
Identify Trust Level
202
Load Symbol Packages Corresponding To Trust Level
204
Lookup External Symbol Reference
206
208
Symbol Found?
N
Y
210
Not Privileged For Operation
205
Get Next External Symbol Reference
212
Resolve External Symbol Reference in Runtime Engine
220
Attempt to Access Resource
216
More Symbols ?
Y
N
222
Retrieve Trust Indicator
218
Execute Entity
224
Trust Indicator Correspond to Resource ?
N
Y
226
Permit Access
Deny Access
228

Fig. 6

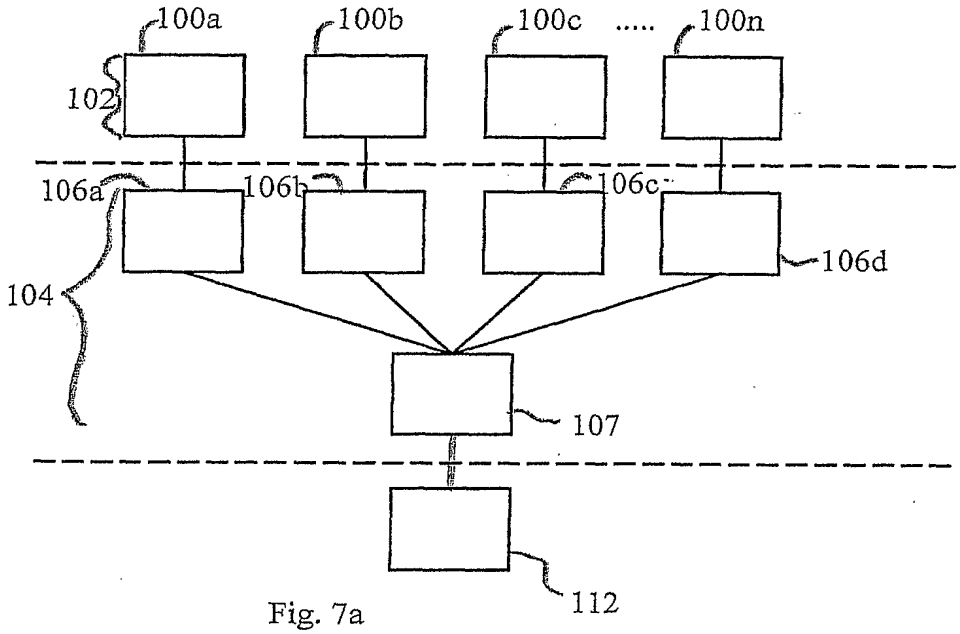

100a
100b
100c
.....
100n
102
106a
106b
106c
106d
104
107
112

Fig. 7a

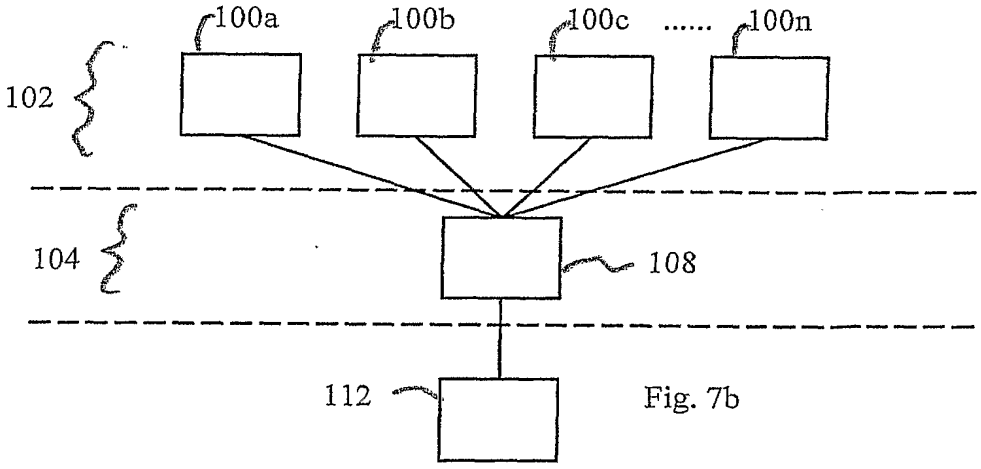

100a
100b
100c
.....
100n
102
104
108
112

Fig. 7b